下载APP

03｜低代码的天花板：一个完备的低代码平台应该具备哪些条件？

2022-03-18 陈旭

《说透低代码》 课程介绍 >

讲述：陈旭

时长 16:08 大小 14.78M

你好，我是陈旭，这一讲我们来探一探低代码的天花板到底在哪儿，也就是从多角度看看，一个完备的低代码平台到底是啥样的。

如果你已经在构建低代码的路上，且自认为干得不错，我想给你描绘一些新的目标，帮助你做得更好，或者至少帮助你了解接下来还能有哪些值得去做的事情。

如果你还未开始构建工作，或者刚刚上路，我想给你一个更加具体的目标，这样你可以更加精确地估计出，如果达成这样的目标，你需要投入多少资源（人力 / 时间）。当然，不是非要触及天花板的低代码平台 / 工具，才有生产力，即使你知悉了天花板在哪，也不定非要去摸一摸，量力而行，适合自己的才是最好的。

这节课，我会从适用领域、适用人员、与基础设施和谐相处、全生命周期这几个角度来说明一个完备的低代码平台的模样。为了避免啰嗦，**我们这一讲讨论的“低代码平台”都特指处于天花板中的低代码平台**。

适用领域

低代码平台需要同时支持以数据为中心和以流程为中心的开发模式。**多**数企业的软件基本上都是围绕着数据或流程打造的，因此对**低代码平台的适用性**提出了极高的要求，要求它能够同时满足多数企业的业务需求的开发。

不同企业的业务复杂程度，显然是不一样的。有的企业的业务复杂度非常高，有的企业则相反，比较简单，但简单的业务往往伴随着数量庞大的特征。

那怎么支撑不同复杂度的业务呢？这一点上，我们要求低代码平台对于简单的业务，需要提供简洁的开发实现方式，并且需要有非常高效的开发效率，从而可以更好地应对简单业务在数量方面的需求。对于复杂度高的业务，低代码平台需要能提供良好的分步实施策略、自然的衔接方式，同时也要充分考虑并提供兜底策略。

在我看来，**兜底策略是应对复杂业务的一个非常有用的方法**。当业务的复杂度超过低代码平台的能力边界，或者业务复杂度过高，使得可视化模式带来的效率 / 易用性收益低于 Pro Code 模式的临界值的时候，低代码平台应该能提供一种方法，允许业务开发人员回退到传统 Pro Code 模式继续开发。并且，这个回退过程应该非常自然 / 内聚、同时尽可能地不影响其他功能的开发。

兜底策略不是一种通用的架构方法，而是一种思路一种策略，它应该在低代码平台各个功能点的架构和开发过程中因地制宜地被实现出来。如果更加极端地运用兜底策略，你会发现我们甚至可以要求低代码平台完全回退到 Pro Code 模式。

实现兜底策略看似难度非常高，但事实并非如此。得益于低代码的关键特征，也就是低代码是有代码的，低代码平台能很好地处理好 Pro Code 模式和可视化开发模式之间的关系。所以即使完全回退到 Pro Code 模式，对平台来说，不但问题不大，反而平台要做的事情会更少。因为在这个模式下，代码是人工产生的，而非低代码平台自动生成的！

当然了，**能做到这点的前提是，低代码平台要有一个足够强大的编译器（代码生成器）**。我会在第 6 讲、第 7 讲中，花整整两讲的篇幅，和你详细说明如何架构、实现这样的编译

器，并帮助你理清编译器和编辑器之间的关系。

虽然一个良好的编译器能让低代码平台具有实现可视化模式到 Pro Code 模式切换回退的基础，但只有这点还不够。

前面我说了，回退的过程应该自然且内聚，这就给 UX 团队的交互设计师和编辑器的开发人员提出了考验。你的交互设计师需要能设计出自然流畅的回退过程，并且编辑器的开发人员要做得出来，并且做出来的性能还要足够好。

最后，在 Pro Code 模式下的低代码平台是一个事实上的 Web Online IDE。没错，也就是说，编码过程智能提示、智能补齐、重构、出错信息及定位等 Native IDE 的功能一个都不能少。

这方面概括地说，就是低代码平台必须具有充分的通用性。

适用人员

说完业务能力的适用性，我们再从低代码平台的用户的角度来看看天花板在哪儿。低代码平台要能**支持各种技能水平的人同时使用**，包括无软件开发技能的业务技术员，也包括掌握某种软件研发专业技能的人，以及水平介于两者之间的各种层次技能人员。

当然，不同层级的人的需求是不同的：

> 业务技术员更需要傻瓜化的操作、更加简洁的操作流程设计，以及“说人话”，意思就是少用编程术语，不得不用时就地给出言简意赅（而非长篇累牍）的说明；
>
> 专业软件研发人员往往要求对他们熟悉领域的底层代码，有更强的掌控能力甚至直接编码，对其他领域则需要有良好的可视化辅助。比如前端人员往往要求有可视化方式可以帮助他们获取业务数据，而后端人员则要求有可视化方式帮助他们画出 UI；
>
> 水平不济的研发人员，特别是职场新手，往往更青睐可视化方式，他们更希望研发全流程都有可视化能力的辅助。

另外，对软件研发人员这类人群，我们还有一个不得不考虑因素：**如何消除他们的职业危机感**。他们往往会非常担心在低代码的绑架下失去职业竞争优势、失去与公司的议价能力。特别是，技能较强的人不仅对这方面的担忧会更加强烈，还会认为低代码让他们失去

了职业经验带来的比较优势，这种情况下，他们会更加抵触低代码技术。这个话题超出了这一讲的内容，但却极现实，影响团队稳定，有机会我们再专门聊聊。

除了个性化要求之外，低代码的各类用户也有共同诉求：

平缓的学习曲线：完善的低代码平台往往可以覆盖业务研发全生命周期的各个环节，所以会包含数量众多的功能、流程。这些对使用人员来说都是知识，在开发流程的设计时，要管控好各种功能对知识的要求和传递，采用渐进式的方式设计开发流程；

尽可能地自动化：没人能抵御自动化的魅力，这也是低代码平台的魔力所在。但在实现各种自动化封装的同时，我们要留出一手，在自动化流程不满足需求时可以切换为手动模式，这就是兜底策略的一种应用。另外，出错时还需给出准确的说明，不能惜字如金，比如出错时弹一个对话框只告知出错的结果，但不包含出错的原因，这样的方式是非常糟糕的；

可以抄作业：也就是说需要提供数量众多的典型应用范例。低代码平台构建初期这个能力很容易被忽视，在平台逐渐成熟之后，我们必须恶补这方面的功课；

提供视频形式的教材：多数人不乐意阅读文字，一份文档有数百字时就嫌烦。视频教材最好采用无音频的外挂字幕方式制作，一方面剪辑起来更加方便，一方面外挂字幕方便搜索和定位；

出问题或者有疑问时能在第一时间得到帮助：建立客服群平台，开发者可以多与使用者互动，热情耐心地帮助他们解决问题，建立首问责任制。

与基础设施和谐相处

这一点主要讨论的是低代码平台与基础设施之间的关系。我认为主要包含两个方面：**一是对运行环境的要求；二是处理好与已有系统，特别是已有数据之间的关系。**

对运行时的要求相对简单。有的低代码厂商可以直接提供基于公有云的运行时，这样就更加方便，基本可以做到开箱即用，但要照顾低代码客户在数据和信息安全方面的担忧。除此之外，我们还可以提供虚拟镜像，并支持在多数基于 PaaS 的平台上安装和运行。这个方式往往用于私有云的部署，这是应对客户信息安全担忧的有效方式。

我认为，低代码平台需要同时对这两种部署方式提供支持，特别是需要支持私有化部署方式。这样，无论是内部使用还是对外售卖，都可以做到更加灵活。当然，那些专注于内部

使用且用户群明确的低代码平台来说，就不需要过多考虑这个问题了。

低代码平台与已有系统之间的关系，大致分为三种：一是被其他系统集成；二是将其他系统集成到平台中来；三是保持相互独立。

虽然，同时支持多种与存量系统融合部署，可以提升低代码平台的适用性，但是我认为没有必要支持所有方式，我们根据自身定位挑选以某种融合方式为主就可以了。

更多的情况下，优先发展被集成的能力是一个更好的选择。创造价值的肯定是业务能力，而低代码平台作为一种开发工具，对多数企业来说，是不具备直接创造价值的能力的，因此低代码平台也就很难冲到前面去了。即使对以售卖低代码能力为生的企业来说，低代码的开发能力是直接创造价值的业务，但一般买家购买后，也是将它作为自身的某个子系统，集成进买家自己的其他业务中。

既然低代码平台一般是被集成、作为子系统使用的，那么低代码平台与存量数据之间的关系也就比较明确了。低代码平台需要提供多种能力来获取外部数据，而不是将自身封闭成一个数据孤岛，要求外部将数据导入到平台中去。而外部数据的结构（关系）、存储介质、数据量、获取渠道等都是无法事先确定的。在这样的前提下，低代码平台要用好这些数据，可想而知难度是非常大的、甚至不具有可行性。

但如果是为确切场景下的数据提供定制化的融合方式，难度会低许多，但是代价也很明显，也就是几乎每种场景都需要提供一份定制。因此我们需要通过适当的架构设计，将定制部分与核心部分解耦，最好是能允许业务团队来自行定制。这正是插件系统需要解决的问题。我在第 15 讲里介绍这一点。

如何做到对单点登录的支持、如何考虑对接权限系统，也是被集成时需要着重考虑的能力。同时还需要在权限系统中区分开发态和运行态，开发态下往往要给予开发人员的账号更大的权限，而运行态下则需要做严格管控，避免数据滥用和渗透。开发态与运行态进行物理隔离，是杜绝开发人员利用开发账号越权访问数据的有效手段，这一点，我们会在接下来的全生命周期里简单阐述。

全生命周期

低代码平台不能只注重开发能力，开发能力当然是低代码平台的关键能力，但绝不是唯一的能力。低代码平台的能力必须**能够覆盖从需求端到应用下线消亡的全过程**。

这中间至少可以覆盖这些环节：

D2C（Design to Code）：业界已经有非常成熟、成功的 D2C 实践案例，对于低代码平台来说，从设计稿中识别出关键信息，再实现与低代码平台的对接与编辑，要比纯 D2C 解决方案容易得多；

UX 设计即开发：有了 D2C 能力后，UX 设计师可以直接提供模板和业务组件。在这个意义下，UX 设计师起到了类似研发人员的作用；

App 开发能力：这点自不必说，这是低代码的重要且基本作用；

App 的自动化测试：包含两点，一是要能帮助 App 自动生成测试代码，二是提供一键式测试环境构建、测试执行、测试报告，乃至自动标注出错位置等；

应用的版本管理：主要体现为要为应用构建相互独立的开发时环境、系统测试时环境、生产环境等，并能实现应用版本的测试、灰度发布、正式上线、紧急回退等能力；

应用生产环境监控：这里包括两点，一是应用运行时基础信息（CPU/ 内存 / 磁盘空间）监控和告警，二是应用埋点数据的植入、采集、分析等。

总之，虽然要做好其中的任何一条就已经很不容易了，但是这所有的功能都应该是低代码平台的“菜”，都可以实现低码化和自动化。第 15 讲我会挑重点对此进行部分讲解，比如 D2C 的实现、零代码生成自动化测试用例等，其他的内容有机会我们再聊聊如何实现。

总结

今天我从多个维度描绘了一个天花板级别的低代码平台必须具有的能力。

从适用领域角度看，低代码平台必须能同时支撑以数据为中心和以流程为中心的 App 的开发，这实际上等于需要覆盖大多数企业的开发业务能力。

同时，它还需要能支持不同复杂度业务的开发、兼顾效率。简单的业务开发要简单高效，而对复杂的业务则需要有良好的兜底策略，确保业务需求突破低代码平台能力边界时，可以有相应的应对措施，比如回退到传统 Pro Code 模式继续开发。这一点又给低代码平台提出了需要支持 Low Code 和 Pro Code 混合开发的方式的要求，而且混合的过程需要具有良好的自然过渡和内聚性。

从适用人员角度看，低代码平台需要能同时支持多种能力背景的人同时使用，比如业务专家群体多数无软件技术技能，需要为他们提供更多傻瓜化、可视化的操作。而且，既要照顾到有软件研发技能的群体在他们专业领域里的诉求，为他们提供更接近底层代码，甚至直接提供编码的开发方式，也要照顾这类群体软件技术能力之外的开发诉求，为他们提供可视化的方式，辅助他们完成业务的开发。

而且，低代码平台要有平缓的学习曲线、尽可能地自动化、提供大量的模板和素材，在帮助他们自学的同时，又能帮助他们解决实际问题。同时，低代码平台也不能只注重开发能力而忽视业务研发生命周期里的其他能力，应该积横向拓展，在需求端和交付端提供能力。

由此可见，要将低代码平台做到天花板上，无论哪个维度上显然难度都非常大，需要投入巨大的资源。但不是非要在触及天花板后，一个低代码平台 / 工具才能产生生产力。我们今天这讲的内容主要是给你描绘出一个完备的低代码平台可以做成啥样，希望能在某些方面对你有所启发。

思考题

在你的场景中，你更看重低代码平台在哪个领域中的表现呢？你希望主要有哪些人来使用你的低代码平台？除了开发能力之外，你还在业务开发过程中的哪些环节上对低代码有需求？

我是陈旭，我们下节课见。

分享给需要的人，Ta订阅超级会员，你最高得 50 元

Ta单独购买本课程，你将得 20 元

生成海报并分享

赞 2 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇
下一篇 04 | 演进策略：先发展通用能力还是先满足业务需求？

精选留言 (2)

写留言

墨白™ 民

2022-03-20

对于大部分公司来说，低代码能发挥的领域肯定不是核心业务逻辑。能够发挥低代码能力的领域，在我看来，主要有以下几种： 1. 营销活动这种快上快下，用完就扔的业务类型适合 2. 面向B端场景下的代码生成，无动画、UI高要求，但对数据处理逻辑这块有通用性的地方，比较适合

展开

作者回复: 对的，同意你的看法。

2

sheeeeep

2022-03-18

「以数据为中心、以流程为中心的开发模式」这两种可以简单的理解为，数据展现类型的业务，和比较多逻辑处理的业务吗？